Method and System for Detecting a Secure State of a Computer System

Field of the invention

[001] The present invention relates generally to computer data security. More particularly, the present invention relates to a method and system of maintaining and evaluating system security to determine whether a malicious computer application such as one of a virus and a Trojan horse application is present in the system.

Background of the invention

[002] There are currently many needs that must be addressed when implementing a system and method for maintaining security within a computer network. One such need is protection from computer code that might compromise the security of the system. An example of this type of program is a "virus," which is a program that loads itself into system memory and then embeds itself into legitimate software so that it can continue to replicate itself. Another example of malicious code is a "Trojan horse," which is a program that appears to be one application such as a simple game or utility but in fact hides its real purpose from the user. Typically a Trojan horse writes itself into the system programming so that it runs every time the computer starts and does not make its presence known to the user. In most cases a Trojan horse is designed for specific attacks on a computer system. Common examples include: opening up access ports to the computer and possibly running system capturing software that allows a hacker to access remotely the computer system by commandeering the mouse, keyboard and screen of the user, or capturing all the keystrokes a user enters and saving them in a file or transmitting them to a location from which the originator of the Trojan horse accesses same.

[003] Although there commonly exists software that scans each application at load time and checks for the existence of viruses in execution, this software typically relies on of the following technique: scanning for known virus code found in an internal database that must be updated frequently.

[004] Unfortunately, most prior art systems do not protect against man-in-the-middle type attacks. A man-in-the-middle attack is a security breach formed when a system or application disposes itself between two parties to a secure communication. For example, a Trojan horse application is interposed between applications in the computer system. The Trojan horse application is for intercepting keysrokes and other security data that are provided between applications within the computer system and for recording or e-mailing the intercepted keystrokes and other security data to the hacker.

[005] In order to prevent locally executing applications, such as for instance a Trojan horse application, from affecting key data, one prior art method is to provide the computer system with a secure token for performing cryptographic functions. The token does not accept programming code and, as such, cannot be tampered with. Further, the token provides no access to keys stored therein, such that security of the key data is assured. Unfortunately, the man-in-the-middle attack is still possible wherein data is intercepted when not encrypted either before being provided to the token or upon receipt therefrom.

[006] Fischer, in U.S. Patent No. 5,436,972 teaches a method for preventing inadvertent betrayal by a trustee of escrowed digital secrets. In a definition phase, the user defines an escrow record that provides self-identification data together with encrypted password data. The user is prompted to voluntarily escrow password or other secret information for later retrieval by entering a series of information uniquely describing them. After unique identification data has been entered, the user is asked to select a password to protect the system. Thereafter, all the personal identifying data, together with the password, is encrypted with the trustee's public key and is stored, for example, in the user's computer as an escrow security record. The password is then used to encrypt all data on the user's disk. If the user forgets the password, a retrieval phase of the method is performed. The trustee utilizes documentary evidence presented by the alleged legitimate user and determines whether such evidence matches with the previously encrypted escrow information stored in the escrow record created by the user. If they agree, then the trustee has confidence that the true owner is making the request, and that revealing the secret key will not betray the owner's interest. Unfortunately, the method disclosed by Fischer does not prevent

a Trojan horse application, when resident in the user's system, from intercepting the private key and recording it.

[007] U.S. Patent No. 5,919,257 issued July 6 1999 in the name of Trostle teaches a networked workstation intrusion detection system. During pre-boot - a period of time prior to initiating operation of the workstation operating system, a networked workstation performs an intrusion detection hashing function on selected workstation executable program(s). A hash function is a mathematical transformation that takes a message of arbitrary length (e.g., the selected executable programs) and computes from it a fixed length number (i.e., the computed hash value). A computed hash value calculated by the hashing operation is compared against a trusted hash value that is downloaded from a server in order to detect unauthorized changes to the selected workstation executable programs. The server can compute the trusted hash value(s) by performing the hashing function on trusted copies of the selected workstation executable programs stored in the server. Alternatively, the server may include a database of trusted hash values each uniquely associated with an executable program.

[008] The workstation executes the hashing function on the selected executable programs stored in workstation memory (disk or RAM) to calculate a computed hash value. The workstation compares this value with the trusted hash value downloaded from the server to determine if any illicit changes have been made to the selected executable programs. If changes are detected, the user and/or the network system administrator is notified in order to take corrective action. Otherwise, the initialization proceeds and the workstation boot process continues. Alternatively, a computed hash value may be calculated separately for each selected workstation executable program, and compared against the trusted hash value downloaded from the server for that executable program. Advantageously, unauthorised changes to executable programs are detected using the prior art method. Unfortunately, it is necessary that the workstation be connected to a server or to other secure hardware from which it can obtain a trusted hash value for comparison. Further, the method compares hash values representing known or authorized executable files, and therefore can only detect Trojan horse applications that are appended to known executable files, and then only if that executable file is included in the comparison.

[009] It would be advantageous to provide a method of reducing the efficacy of a man-in-the-middle security attack.

Object of the Invention

[0010] It is an object of the invention to provide a method and system for detecting unauthorized applications prior to providing security data from a trusted store.

[0011] It is an object of the invention to provide a method and system for detecting locally executing Trojan horse applications and viruses prior to providing security data from a secure source.

Summary Of The Invention

[0012] In accordance with the invention there is provided a method of detecting unauthorized executable programs resident in a computer system memory comprising the steps of:

a) receiving a trusted hash value representative of a hash value for generation by a predetermined hashing process of predetermined data stored in memory within the computer system if an unauthorized executable program is other than resident in the computer system;

b) hashing the data stored in memory within the computer system using the predetermined hashing process to determine a computed hash value; and

c) comparing the computed hash value and the trusted hash value to determine differences between the data and the predetermined data.

[0013] In accordance with an embodiment of the instant invention there is provided is a method of detecting unauthorized executable programs resident in a computer system comprising the steps of:

a) providing a trusted security application executable on a processor of the computer system for determining a hash value using a predetermined hashing process of predetermined data existing in memory within the computer system;

b) hashing the data existing in memory within the computer system using the predetermined process to determine a hash value;

c) digitally signing the hash value to provide a trusted hash value; and

d) retrievably storing the trusted hash value,

wherein the hash value is determined absent an unauthorized executable program being present within the computer system; and

wherein the predetermined data relates to programs in execution on the processor of the computer system.

Brief Description Of The Drawings

[0014] The invention will be readily understood by the following description of preferred embodiments, in conjunction with the following drawings, in which:

[0015] **Figure 1a** shows a simplified block diagram of a prior art computer system for use with the instant invention;

[0016] **Figure 1b** shows a simplified block diagram of a prior art networked computer system for use with the instant invention;

[0017] **Figure 2** shows a simplified code block diagram according to a first embodiment of the instant invention for use with the computer system of Figure 1a;

[0018] **Figure 3** shows a simplified flow diagram of a method for generating a trusted hash value according to the first embodiment of the instant invention;

[0019] **Figure 4** shows a simplified flow diagram of a method for detecting a malicious computer application according to the first embodiment of the instant invention

[0020] **Figure 5a** shows a simplified code block diagram according to a second embodiment of the instant invention for execution on the end-user computer system shown in Figure 1b;

[0021] **Figure 5b** shows a simplified code block diagram according to the second embodiment of the instant invention for execution on the secure server shown in Figure 1b;

[0022] **Figure 6** shows a simplified flow diagram of a method for detecting a malicious computer application according to the second embodiment of the instant invention.

Detailed Description of the Invention

[0023] The instant invention is concerned with an implementation of a system, using trusted secure software applications, for the purpose of maintaining security within one of a computer system and a network of computer systems, such that an end user can run un-trusted applications. According to the instant invention, a secure state of a computer is verified in an initial state and the verification data is then used to assure the secure state of the computer. A trusted hash value, which is indicative of the secure state of the computer system, is generated, digitally signed and retrievably stored for later use in assessing a current state of the computer system.

[0024] Referring to Figure 1a, shown is a simplified block diagram of a prior art computer system **11**. The computer system **11** includes a processor **12** in electrical communication with each of a memory storage device **13** and a volatile memory circuit **34**. Memory storage device **13** in the form of an internal disk drive, is for storing program code that is executable on processor **12**, for example a word processor application and a database application. Of course, related memory management files, such as for instance dynamic linked library (DLL) files, are also stored within memory storage device **13**. Additionally, data that has been processed by the at least an application is stored within memory storage device **13**. Volatile memory circuit **34**, for instance a random access memory (RAM) bank, is for storing program code and related memory management files during execution of the code on processor **12**. Processor **12** is also in electrical communication with an authentication information input device **14**, such as for instance a fingerprint scanner, via an interface **15**. As will be obvious to one of skill in the art, input device **14** is optionally one of a plurality of well known other authentication input devices, including: a keyboard, a retinal scanner, a voice recognition system and a magnetic card reader.

[0025] Referring now to Figure 1b, shown is a simplified block diagram of a prior art client/server system **10** comprising a computer system **9** in electrical communication with a secure server **17** via a communication network **21**. Those elements that are identical to elements of Figure 1a have been assigned identical reference numerals, and their discussion has been omitted for the sake of brevity. The computer system **9** includes an input/output port **16** in electrical communication with processor **12** for exchanging data with the secure server **17** via the communication network **21**. The secure server **17** includes a processor **18** in electrical communication with a port **20** and with a memory storage area **19**. A communication path for exchanging data between the computer system **9** and the secure server **17** includes the elements **16**, **21** and **20**. Optionally, the ports **16** and **20** are secure input/output ports.

[0026] Referring now to figure 2, shown is a code block diagram according to a first embodiment of the instant invention. Figure 2 depicts a group of applications **26** for execution on processor **12** of the prior art system **11** of Figure 1a. Group **26** includes a "trusted group" of applications **4** and at least an un-trusted end application **1**, such as for example a word processor application. The un-trusted application **1** includes code for performing user authorization using, for example, a standard password authorization system **3**. In use, a user of system **11** initiates an action requiring a password, such as for instance attempting to access a user data file **2** associated with the un-trusted application **1**. The un-trusted application **1** prompts for a password, which is detected by the trusted group of applications **4**. The trusted group of applications **4** includes a secure user-authorization system **6**, which prompts the user to provide user authorization information via the input device **14**. The secure user-authorization system **6** compares the provided sample with a template retrieved from a user verification database **7** and verifies the identity of the user in dependence upon the comparison. Once the identity of the user is verified, a hash generator **22** accesses data **8** stored in one of the memory storage device **13** and the memory circuit **34** of computer system **11**. Using a predetermined hashing algorithm, hash generator **22** determines a unique hash signature for the current state of the applications running within the computer system **11**. For example, the hash generator **22** comprises executable code for examining data of the operating system memory such as the DLL tables or the system call stack, and for generating a hash value representative of the examined data. A hash verifier **23** compares the generated system hash value to a trusted hash value **5**, the trusted

hash value **5** having been generated previously and retrievably stored when the computer system **11** was in a verified secure state. The trusted hash value is encoded to ensure that it is not tampered with. For example, the trusted hash value is digitally signed by a trusted entity such as the secure-user authorization system **6**. Hash verifier **23** includes code for decrypting the encrypted hash value and for verifying a digital signature of same. When the comparison performed by hash verifier **23** indicates that the generated hash value is identical to the trusted hash value **5** and the trusted hash value is not tampered with, a password provider **24** of the trusted group of applications **4** accesses a password database **25** in order to retrieve the requested password. The password provider **24** provides the retrieved password to the password authorization system **3** of un-trusted application **1**, thereby allowing the user to access and decrypt the user data file **2**.

[0027] Referring now to Figure 3, shown is a simplified flow diagram of a method according to the present invention for obtaining a trusted system hash value when the computer system **11** is in a known or initial state, in particular a secure state. At step **100** the secure user-authorization system **6** verifies the identity of a system administrator, the system administrator being authorized to establish a trusted hash value of the computer system **11**. At step **101** the system administrator provides a command to the trusted group of applications **4** indicative of a secure state of the computer system **11**. The hash generator **22** examines at step **102** data **8** stored in one of the memory storage device **13** and the memory circuit **34** of computer system **11**. Using a predetermined hashing algorithm, hash generator **22** determines a trusted hash value for the current trusted state of the applications running within the computer system **11**. For example, the hash generator **22** comprises executable code for examining data of the operating system memory such as the DLL tables or the system call stack, and for generating a hash value representative of the examined data. Of course, other operating state independent hash values are also determinable. At step **103** the hash generator **22** digitally signs through a step of encryption the trusted hash value thus obtained, for instance using a private key that is other than available outside of by the system administrator using the trusted group of applications **4**. The digitally signed trusted hash value is retrievably stored by the hash generator **22** at step **104**, and the method of Figure 3 terminates at step **105**.

[0028] Advantageously, the authenticity of the digitally signed trusted hash value is verifiable by hash verifier **23** when it is decrypted. Further advantageously, the trusted hash value is stored in an encrypted form that can only be reproduced by the hash generator of the trusted system using a private key that is other than accessible outside of the trusted group of applications **4**. As such, it is very difficult for a hacker to fake the digitally signed and encrypted trusted hash value.

[0029] Of course, though a detailed method of generating the hash value is described, once generated for a known installed system, a same hash value is usable by identically configured systems. For example, a company installing identical computer systems on a plurality of desktops could generate a system image for installation on each computer. Since the system image is trusted and identical for different computers, the trusted hash value will also be identical. Thus, it is possible to generate the trusted hash value in a secure environment having physical security as well and then to merely install it on user systems. This obviates a need for additional private key security for the private key used to digitally sign the hash value.

[0030] Referring now to Figure 4, shown is a flow diagram of a method according to the first preferred embodiment of the instant invention. The method is initiated at step **110** when the un-trusted application **1** transmits to the trusted group of applications **4** a request for a password, for instance the request is transmitted in response to an attempt by the user to access an encrypted user data file **2**. Often such a request is transmitted by a trap process in execution on the system for detecting password request operations. As is evident tot hose of skill in the art, it is difficult to ensure that each version of each application in execution on a system is secure. Further, it is difficult to ensure that a password provided to an application is for the "anticipated" use.

[0031] At step **111** the secure user-authorization system **6** prompts the user to provide user authorization information, for instance a biometric information sample provided via input device **14**. At decision step **112** the secure user-authorization system **6** compares the provided information with template information retrieved from the user verification database **7** and determines if there is a match within predetermined limits. For example, the user authorization information is a fingerprint image and fingerprint recognition processes are employed to identify and/or authorize the user. When the comparison indicates other than a match, the trusted group

of applications **4** refuses at step **119** the password request and the method of Figure 4 terminates at step **118**. Optionally, the trusted group of applications **4** transmits a message indicative of an unauthorized access attempt to a system administrator of the computer system. Further optionally, the password provider **24** transmits a blank and thus incorrect password to the password authorization system **3** of the un-trusted application **1**, thereby denying the user access to said data file **2**.

[0032] When the comparison at decision step **112** indicates a match, the hash generator **22** accesses data **8** stored in at least one of the non-volatile memory **13** and the volatile memory **34** of computer system **11**. Using a predetermined hashing algorithm, hash generator **22** at step **113** determines a unique hash signature for the current state of the applications running within the computer system **11**. For example, the hash generator **22** comprises executable code for examining data of the operating system memory such as the DLL tables or the system call stack, and for generating a hash value representative of the examined data. At step **114** the hash verifier **23** retrieves the digitally signed trusted hash value for the computer system from memory and decrypts the trusted hash value using an associated public key. If at decision step **115** the digital signature is determined to be authentic, the hash verifier **23** at decision step **116** compares the hash value generated at step **113** with the trusted hash value retrieved at step **114**. If the two hash values match within predetermined limits, the password provider **24** accesses the password database **25**, retrieves the requested password and at step **117** transmits the retrieved password to the password authorization system **3** of un-trusted application **1**. Typically, the hash values must match. The method of Figure 4 terminates at step **118**.

[0033] Of course, if at decision step **115** the digital signature is determined to be other than authentic, or if at decision step **116** the two hash values are determined to be other than identical, the trusted group of applications **4** refuses the password request at step **119** and the method of Figure 4 terminates at step **118**. Optionally, the password provider **24** transmits a blank and thus incorrect password to the password authorization method **3** of the un-trusted application **1**, thereby denying the user access to said data file **2**. Further optionally, the trusted group of applications **4** locks-out future access from all un-trusted applications in execution on processor **12** when at step **116** it is determined that the two hash values are other than identical. Locking-

out access prevents the trusted applications from providing passwords to maintain the passwords in a secure environment. The purpose of preventing the provision of password data in a situation wherein the system state has been previously compromised is to prevent a virus or a Trojan horse application from intercepting and recording the password data that is passed between the secure group of applications **4** and the unsecure application. Furthermore, since the system has been compromised already, it is redundant to continue re-running the secure user-authorization system **6** until the system has been restored to a known secure state.

[0034] Referring now to figures 5a and 5b, shown are code block diagrams according to a second embodiment of the instant invention. This embodiment is similar to the first embodiment discussed supra, however, it differs in that the trusted group of applications is distributed across multiple platforms, for instance the end-user computer system **9** and the secure server **17** of the networked system **10** shown in Figure 1b. Figure 5a depicts a group of applications **50** for execution on processor **12** of the computer system **9**. Group **50** includes a "trusted group" of applications **51** and at least an un-trusted end application **1**, such as for instance a word processor application. The un-trusted application **1** includes code for performing user authorization using, for example, a standard password authorization system **3**. In use, a user of the networked system **10** initiates an action requiring a password, such as for instance attempting to access a user data file **2** associated with the un-trusted application **1**. The un-trusted application **1** transmits a request for a password to the trusted group of applications **51**. The trusted group of applications **51** includes a secure user-authorization system **6**, which prompts the user to provide user authorization information via the input device **14**. The secure user-authorization system **6** compares the provided sample with a template retrieved from a user ID verification database **7** and verifies the identity of the user in dependence upon the comparison. Once the identity of the user is verified, a hash generator **22** accesses application data **8** stored in one of the memory storage device **13** and the memory circuit **34** of computer system **9**. Using a predetermined hashing algorithm, hash generator **22** determines a unique hash signature for the current state of the applications running within the computer system **9**. For example, the hash generator **22** comprises executable code for examining data of the operating system memory such as the DLL tables or the system call stack, and for generating a hash value **35** representative of the examined data. The system transmits the generated hash **35** value using a secure send driver **34** of port **16**,

which includes a predetermined secure transmission protocol. For example, the transmission is made over a known secure transmission route, via a tunnel, or using known session encryption techniques. This allows the generated hash value **35** to be transmitted over a network **21** to be received by a secure server **17** via port **20**.

[0035] Figure 5b depicts a group of applications **37** for execution on processor **18** of the secure server **17**. Group **37** includes a "trusted group" of applications **66**. In use, the secure server **17** uses a secure receive driver **38** of port **20** to receive the hash value **35** transmitted from the trusted group of applications **4** of computer system **9**. The hash value **35** is passed to the hash verifier **23**. The hash verifier **23** retrieves a trusted hash value **5** for computer system **9** that is stored locally on secure server **17**. For instance, the trusted hash value is generated when computer system **9** is in a known secure state according to the method of Figure 3 and transmitted via network **21** for storage in memory storage area **19** of secure server **17**. The trusted hash value is typically stored in an encoded form to ensure that it is not tampered with. For example, the trusted hash value is digitally signed by a trusted entity such as the secure-user authorization system **6** of computer system **9**. The hash verifier **23** verifies the digital signature of the trusted hash value **5** and compares the transmitted hash value **35** to the trusted hash value **5**. When the comparison indicates that the generated hash value **35** is identical to the trusted hash value **5**, thus confirming the secure state of the remote computer system **9**, a password provider **24** of the trusted group of applications **66** of the secure server **17** accesses a password database **25** in order to retrieve the requested password. The password provider **24** provides the retrieved password **40** in encrypted form to the un-trusted application **1** of computer system **9** using a secure send driver **39** of port **20**. A secure receive driver **41** of port **16** receives the encrypted password **40** and provides same to password authorization system **3** of the un-trusted application **1**, thereby allowing the user of computer system **9** to access the user data file **2**.

[0036] Referring now to Figure 6, shown is a flow diagram of a method according to the second embodiment of the instant invention. The method is initiated at step **110** when the un-trusted application **1** transmits to the trusted group of applications **51** a request for a password, for instance the request is transmitted in response to an attempt by the user to access an encrypted user data file **2**. Often such a request is transmitted by a trap process in execution on

the system for detecting password request operations. At step **111** the secure user-authorization system **6** prompts the user to provide user authorization information, for instance a biometric information sample provided via input device **14**. At decision step **112** the secure user-authorization system **6** compares the user-provided information with template information retrieved from the user verification database **7** and determines if there is a match within predetermined limits. For example, the user authorization information is a fingerprint image and fingerprint recognition processes are employed to identify and/or authorize the user. When the comparison indicates other than a match, the trusted group of applications **51** refuses at step **131** the password request and the method of Figure 6 terminates at step **130**. Optionally, the trusted group of applications **51** transmits a message indicative of an unauthorized access attempt to an administrator of the computer system.

[0037] When the comparison at decision step **112** indicates a match, the hash generator **22** accesses application data **8** stored in one of the memory storage device **13** and the memory circuit **34** of computer system **9**. Using a predetermined hashing algorithm, hash generator **22** determines a unique hash signature for the current state of the applications running within the computer system **9**. For example, the hash generator **22** comprises executable code for examining data of the operating system memory such as the DLL tables or the system call stack, and for generating a hash value **35** representative of the examined data. At step **120** the generated hash value is transmitted to the secure server **17** via the network **21**, for instance using a secure send driver **34** of the input/output port **16** which prepares a request and which optionally encrypts the generated hash value prior to transmission.

[0038] The secure server **17** receives at step **121** the transmitted hash and request via a secure receiving driver **38** of input/output port **20**. Of course, when the hash value is encrypted prior to transmission, step **121** includes the optional step of decrypting the transmitted data. At step **122** the hash verifier **23** retrieves the digitally signed trusted hash value **5** from memory **19** and decrypts the trusted hash value using an associated public key. If at decision step **123** the digital signature is determined to be authentic, then the hash verifier **23** compares the generated hash value to the trusted hash value at decision step **124**. If the comparison indicates that the two hash values are identical, the password provider **24** accesses the password database **25** and

retrieves at step **125** the password. The step of retrieving the password includes the step of encrypting the password using a predetermined private key to provide an encrypted password **40**. Optionally, the step of retrieving the password includes the step of retrieving an encrypted password **40** from the password database **25**. At step **127** the encrypted password **40** is transmitted using a secure send driver **39** of input/output port **20**. At step **128** the encrypted password **40** is received using a secure receive driver of input/output port **16** of computer system **9** and is decrypted within the secure group of applications **51**. The requested password is provided in decrypted form to the password authorization system **3** of the un-trusted application **1** at step **129**. The method of Figure 6 terminates at step **130**.

[0039] Of course, if at decision step **123** the digital signature is determined to be other than authentic, or if at decision step **124** the two hash values are determined to be other than identical, the trusted group of applications **66** refuses at step **126** the password request and the method of Figure 6 terminates at step **130**. Optionally, the password provider **24** transmits a blank and thus incorrect password to the password authorization method **3** of the un-trusted application **1**, thereby denying the user access to said data file **2**. Further optionally, the trusted group of applications **66** sends a lock command to the end user trusted group of applications **51** to lock-out future access from any un-trusted application when the decision at step **124** is that the hash values do not match. Locking-out access to the trusted group of applications **51** prevents the user from providing user authorization information via input device **14**. The purpose of preventing the input of user authorization information in a situation wherein the system state has been previously compromised is to prevent a virus or a Trojan horse application from intercepting and recording or otherwise modifying the data passed between the input device **14** and the secure group of applications **51**. Furthermore, since the system has been compromised already, it is redundant to continue re-running the secure user-authorization system **6** until the system has been restored to a known secure state.

[0040] Optionally, when the computed hash value and the trusted hash value are other than a same hash value, the methods of Figures 4 and 6 initiate a routine wherein the authorized user of the computer system is prompted to verify that a detected unauthorized executable program is an unauthorized program installed from a secure source. Of course, the user must possess an

acceptable security clearance as indicated in the ID verification database **7** to be allowed to override the security application of the instant invention. When the detected unauthorized executable program is verifiably secure, the security application provides the requested encryption data from the trusted source. Further optionally, a new trusted hash value reflecting the unauthorized changes to the system is prepared, for instance according to the method of Figure 3.

[0041] Of course, numerous other embodiments may be envisaged without departing significantly from the spirit or scope of the invention.